Achieving High Software Reliability Using a Faster, Easier, and Cheaper Method

Prepared for

NASA Independent Verification and Validation Facility

FAU Technical Report TR-CSE-01-20

Taghi M. Khoshgoftaar*
Linda Lim
Erik Geleyn

Florida Atlantic University
Boca Raton, Florida USA

July 2001

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu, URL: www.cse.fau.edu/esel/.

Executive Summary

The increased reliance on software driven systems requires an increase in software quality. Poor software quality can threaten safety, put a company at risk, or alienate potential customers [4]. With more lives being impacted by software-driven devices, it is important to ensure that these devices are reliable and accurate.

An effective way to ensure software reliability is to apply one of the various classification or prediction modeling techniques to software quality data. However, published modeling methods do not always produce software quality models with sufficient accuracy. Thus, more accurate and robust modeling methods are needed.

Our research demonstrates how software quality modeling methods like case-based reasoning can be effective in facilitating the decision-making processes for project managers so they can achieve the all-important goals of releasing a product on time and within the allocated budget, all the while adhering to established quality standards. There are various techniques that can be used to build software quality models. There are also many different types of software quality models that can be built. They include fault prediction and fault classification models.

Case-based reasoning (CBR) is an alternative software quality modeling method based on automated reasoning processes. It has proven useful in a wide variety of domains. CBR is especially useful when there is limited knowledge about the process and when an optimal process solution is not known. However, to our knowledge very few CBR systems for software quality modeling have been deployed.

A CBR system finds a solution to a new problem based on past experience, represented by cases in a case library. Each case is indexed for quick retrieval according to the problem domain. A solution process algorithm uses a similarity function to determine the most similar modules from the case library to a target module with respect to the independent variables. The algorithm retrieves relevant cases and determines a solution to the target problem. A CBR system can function as a software quality classification model. The objective is to assign a module to the correct class or to a prediction of a quality factor early in development. A good "solution" is a prediction that turns out to be correct after fault data is known.

While previous work [2] didn't provide extensive empirical work, this research presents empirical results with CBR using two case studies. Software metrics data for both the case studies was obtained as a result of an extensive data collection effort performed at Motorola Inc. Through the empirical work with these case studies, we attempt to build useful software quality estimation models to predict the quality of the software prior to system tests, and by analyzing the fault removal process, determine where time, money, and energy can be saved. The first case study sought to determine the quality of the software just prior to the system test by building classification and prediction models. The second case study sought to analyze the fault removal process during the system tests to determine whether or not an inspection for a fault-fix is necessary. Our second case study proves that by analyzing several attributes, quality models can be built that may determine the outcome of an inspection as accept or reject. By not having to inspect all fixes, the development team can redirect efforts to fixing the more complex faults or to increasing the number of faults fixed prior to market release.

From this research, we have proven that CBR is a simple yet very effective methodology for software quality modeling. We have also shown that we could build useful software quality models using very few metrics and yield good results.

Keywords: software reliability, faults, fault-prone modules, software metrics, classification, prediction, return on investment, case-based reasoning.

1 Introduction

Recent publications have stated that models and measurements were the means for understanding, controlling, and improving development processes [19] in the software engineering field. Prior applied research sponsored by NASA have summarized methods for modeling fault-prone software modules [5], and the benefits of applying a case-based reasoning approach [3].

Software product metrics are quantitative attributes of software abstractions. We can consider three main types of software product metrics: call graphs, control flow graphs, and statements. Attributes of a node in a call graph like fan-in and fan-out are examples of call graph metrics [18]. Control flow graph metrics are more commonly used and McCabe's cyclomatic complexity is one of the best known in this category [17]. Statement metrics are really common and the most famous is surely the number of lines of code, but number of operators or operands are other good examples of statement metrics.

Software process metrics refer to the process development and can be retrieved from problem reporting systems and configuration management systems. Process metrics turn out to be especially useful for large legacy systems with high reuse.

Various classification and prediction modeling techniques can use these software metrics as input to compute respectively either a class membership (fault-prone, not fault-prone) or an estimation of a quality factor (number of faults in the module under development). Previous research at the Empirical Software Engineering Laboratory (ESEL) at Florida Atlantic University (FAU) has used discriminant analysis [4], logistic regression [6], decision trees [7][20], artificial neural networks [8][9], and fuzzy logic [23].

An effective prediction and classification modeling technique is case-based reasoning. Based on the assumption that a future module will have the same number of faults as a similar module developed previously, a case-based reasoning model is able to retrieve a prediction from past instances, from cases in the CBR system library [3]. These past instances are the *cases* of our library. They are well-known data from past projects. These *cases* contain all the available information about the described program module, and include:

1. The independent variables to evaluate the similarities between the modules.

2. The dependent variable to make a prediction for future program modules.

In our case study we wanted to estimate a quality factor or a class (fault-prone, not fault-prone) in a currently developed program module, to be able to take corrective action early in the life cycle. Case-based reasoning models use *similarity functions* to determine from the case library the most similar items to the module under development. Various *similarity functions* are available and they all compute a distance from the studied module to the cases in the library. The closest modules determined by the *similarity function* are called *nearest neighbors*. We can then select the number of *nearest neighbors* we want to consider to build the prediction for the dependent variable using a *solution algorithm*. *Solution algorithms* compute a prediction of the dependent variable using the dependent variables of the *nearest neighbors* from the case library. Some algorithm give the same importance to all the *nearest neighbors* while others give more importance to the closest cases.

Our research is primarily focused on two case studies. Through these case studies, we attempt to build useful software quality models in order to predict the quality of the software prior to system test and, by analyzing the fault removal process, determine where time, money, and energy can be saved. We used few primitive software metrics and yielded good results. The first case study sought to determine the quality of the software just prior to system test. In this case study, we built classification models and prediction models. The classification models classified source files as either fault-prone or not fault-prone based upon a threshold determined by the user. A source file represents a collection of high-level code modules on a per feature basis. Experiments were conducted by varying the user-defined thresholds and model validation techniques. The prediction models predicted the number of faults discovered within each source file during system test. The second case study sought to analyze the fault removal process during system test to determine whether or not an inspection for a fault-fix is necessary.

The objectives pursued can be enumerated in the following way:

1. We would like to classify the fault-proneness of source files and predict the number of faults discovered during system test to predict the quality of the software prior to system test. Predicting the software quality early will provide managers with the opportunity to take appropriate software quality enhancement strategies.

2. We would like to prove that by analyzing the fault removal process, managers could save time, money, and energy by determining which inspections are worth conducting. By classifying the outcome of an inspection as accept versus reject, managers can save resources by inspecting only the fault-fixes that the model deems necessary.

From this research, we have proven that through the use of a simple yet very effective methodology, CBR, we can build useful software quality models using very few primitive metrics and yield good results. The Type I and Type II misclassification rates were used to evaluate the classification accuracy of the software quality classification models. Where a Type I error misclassifies a not fault-prone module as fault-prone, and a Type II error misclassifies a fault-prone module as not fault-prone. The average absolute error and the average relative error values, were used to evaluate the predictive accuracy of the software quality prediction models.

The remainder of this paper is structured as follows: Section 2 details the CBR-based modeling methodology implemented in this research. Section 3 and 4 reveal the experiments conducted in both case studies, and the results obtained from these experiments. Section 5 explains the conclusions drawn from this study and identifies possible future work in this area of research.

2 Case-Based Reasoning

Case-Based Reasoning is a type of modeling methodology that is based on automated reasoning processes [10]. It is an attractive method to implement because it is based on intuitive human reasoning. As a result, CBR methods are very easy to use, and the results are easy to understand and interpret. CBR has been used successfully in areas such as software cost estimation, software reuse, and software design [16]. Our research concentrates efforts on demonstrating the use of CBR to develop useful and accurate software quality models.

In addition to its simplicity and ease of use, CBR is very flexible because it is applicable for both classification and prediction modeling. In our research, we capitalize on this flexibility by generating several software quality models for both prediction and classification. The additional advantages of using CBR over other modeling techniques include [11]:

1. The ability to alert users when a new case is outside the bounds of current experience.
2. The ability to interpret automated classification through the detailed description of the most similar case.
3. The ability to take advantage of new or revised information as it becomes available.
4. The ability for fast retrieval as the size of the library scales up. CBR is scalable to very large case libraries and is amenable to concurrent retrieval techniques.

The main premise of the CBR method is to look at past cases that are similar to the present case in an attempt to predict or classify the specific attributes desired. For instance, assume that a CBR system functions as a software quality classification model to classify current cases as fault-prone versus not fault-prone. The following can then be described as the working hypothesis: current cases that are in development will more than likely also be fault-prone if past cases having similar attributes were fault-prone [10]. Determining whether a case is considered fault-prone is quite subjective as it is often based upon whether or not it exceeds a particular threshold set by the user

The past cases are data that were collected from previous projects. These cases are stored in a library, and each case contains all the known attributes. These attributes provide descriptive information about each case. The attributes can be further categorized into dependent variables and independent variables. The independent variables are variables that are known early in the software development cycle. The dependent variables, on the other hand, are not known until the later phases of the software development cycle. These are the variables that we would like to classify or predict by analyzing the set of independent variables that are known early.

The past cases contained in the library are known collectively as the fit data set. In the fit data set, all the attributes (independent variables and dependent variables) are known, and a model is built from this information. The target data (test data) set containing the present cases are often from a current project. In the target data set, the independent variables are the only attributes that are known. By applying the model built using the fit data set, we can classify or predict the dependent variable(s) desired in the current project early in the life cycle. Management can utilize this information for planning, decision making, and quality assessment purposes.

If there is only one data set available, then strategies can be implemented to obtain a target data set for model validation. This is the situation that was encountered in our research. There are two solutions that can be implemented when only one data set is available for software quality modeling. They include cross-validation and data splitting. Data splitting derives the fit data set and target data set by randomly sampling from the cases available and impartially partitioning them into the two data sets. Cross-validation is described [12] in the following manner:

- Suppose there are n observations available in a single data set. Let one observation be the target data set and all others be the fit data set.

- Build a model and evaluate it for the current observation or target data set.

- Repeat for each observation, resulting in n models.

- Let the misclassification rates summarize the n evaluations of the models.

For our research, we used both the cross-validation method and the data splitting method to build and validate our software quality models.

To find past cases from the case library that are similar to the present case, CBR uses a similarity function. A similarity function measures the relationship or similarity of the present case to every other case in the case library. The relationship is determined by a distance measure. The closer a present case is to a case in the case library, the more similar it is to that particular case. The cases that are most similar to the present case are noted, and from this set of cases, a solution can be derived. There are several types of similarity functions available including Absolute Distance, Euclidean Distance, and Mahalonobis Distance. Based upon previous research [21][22], the Mahalonobis Distance similarity function revealed higher performance accuracy with raw data sets than the other similarity functions for both prediction and classification models.

For classification models, classification methods are selected. Classification methods include Majority Voting and Data Clustering. Based upon previous research [21], Data Clustering was shown to be a better classification method.

If CBR is being used to build software quality prediction models, then in addition to selecting a similarity function, a solution process algorithm is also selected. The solution process algorithm uses the similarity function to estimate the actual value of the dependent variable(s). Types of solution algorithms are Unweighted Average and Inverse-Distance Weighted Average. Based upon previous research [22], the Inverse-Distance Weighted Average performs better than the Unweighted Average.

In summary, the general steps for CBR modeling methodology are as follows:

1. A fit data set is specified as the case library. This data set represents data from a past project where all the independent and dependent variables are known. This information is used to build the software quality models. Let c_{jk} be the value of the k^{th} independent variable for case j and let c_j be the vector of independent variable values for case j [10].

2. A target data set is specified for model validation. This data set represents data from a current project where the dependent variables are unknown. The dependent variables are the attributes that are to be classified or predicted by the models built. Let x_{ik} be the value of the k^{th} independent variable for target case i, and let x_i be the vector of independent variable values for case i [10].

3. A similarity function is selected. The similarity function calculates the distance, d_{ij}, between the current case x_i and every case c_j in the case library. A small distance indicates that the cases are similar.

4. The number of cases to be used in the solution algorithm is selected (within the nearest neighbors). Once the distances are computed for every case c_j, they are then sorted. The cases with the smallest distances are of primary interest. Let N be the complete set of nearest neighbors, which is the set of cases contained within the fit data set that are most similar to the current case, x_i. The number of cases, n_N, represents the number to be selected out of N for analysis and prediction. During the model development, the user can experiment with various values in order to reach an optimum number for the model.

5. For prediction models, the solution algorithm is selected. The solution algorithm

will estimate the actual value of the dependent variable(s) for the current case, x_i.

6. For classification models, the classification method is selected (data clustering or majority voting).

7. The models are evaluated for accuracy. For prediction models, the evaluation is performed by assessing the error of the predicted dependent variable. The two error values that are calculated include Average Absolute Error and Average Relative Error. For classification models, the evaluation is performed by assessing the Type I and Type II misclassification rates. These rates are discussed in greater detail later.

2.1 Similarity Function

A similarity function is used to calculate the distance, d_{ij}, from the current case, x_i, to each of the cases in the case library, c_j. It should be noted that the raw metrics collected, i.e., the independent variables, are often measured in varying ways and could contain a wide variety of ranges and scales. With this being the case, it is often beneficial to first standardize the metrics. "Standardization" is a method that allows all metrics to use the same unit of measure. For each metric, x_i, the standardized metric is computed according to the following formula:

$$Z_i = \frac{x_i - \overline{x}_i}{S_i} \qquad (1)$$

where, $\overline{x}_i$ is the mean and S_i is the standard deviation of the i^{th} metric, x_i. Standardization is not necessary for all of the similarity functions available. In fact, the Mahalonobis

Distance (described later in this section) similarity function does not require standardization. The following are some similarity functions that can be used in CBR for both prediction and classification.

Absolute Difference Distance: This distance is also known as City-Block Distance or Manhattan Distance. It is calculated by taking the weighted sum of the absolute value of the difference in independent variables between the current case and a past case. The user of the model provides the weights, and the absolute value is taken because the direction of the difference in distance is irrelevant. This distance is primarily used for numeric attributes. The following is the equation for Absolute Difference Distance:

$$d_{ij} = \sum_{k=1}^{m} w_k |c_{ij} - x_{ik}| \tag{2}$$

where, m is the number of independent variables, and w_k is the weight of the k^{th} independent variable.

Euclidean Distance: This distance views the independent variables as dimensions within an m-dimensional space, with m being the number of independent variables. A current case is represented as a point within this space. The distance is calculated by taking the weighted distance between a current case and a past case within this space. Again, the user of the model provides the weights, and this distance is also commonly used when the data set contains quantitative attributes. The following is the equation for Euclidean Distance:

$$d_{ij} = (\sum_{k=1}^{m} (w_k(c_{jk} - x_{ik}))^2)^{\frac{1}{2}} \tag{3}$$

Mahalonobis Distance: This distance measure is an alternative to the Euclidean Distance. It is used when the independent variables are highly correlated. Mahalonobis Distance is a very attractive similarity function to implement because it can explicitly account for the correlation among the attributes, and the independent variables do not need to be standardized. In cases where the variances of the independent variables have unit variances and are uncorrelated, the Mahalonobis Distance is simply the Euclidean Distance squared. The following is the equation for Mahalonobis Distance:

$$d_{ij} = (c_j - x_i)' S^{-1} (c_j - x_i) \tag{4}$$

Prime ($\prime$) means transpose, and S is the variance-covariance matrix of the independent variables over the entire case library. S^{-1} is its inverse.

2.2 Solution Algorithms

A solution algorithm is used in prediction models to estimate the actual value of the dependent variable(s). The following are some solution algorithms that can be used in CBR for prediction modeling.

Unweighted Average: This algorithm estimates the value of the dependent variable(s), $\widehat{y}_i$, for the current case by taking the average of the dependent variable(s) of the closest n_N cases from the case library. The following is the equation for Unweighted Average:

$$\widehat{y}_i = \frac{1}{n_N} \sum_{j \in N} y_i \tag{5}$$

Inverse-Distance Weighted Average: This algorithm uses the distance measures between the current case and the closest cases in the case library as weights in a weighted average. Because a smaller distance indicates a better match, each case in the nearest neighbors set is weighted by a normalized inverse distance, δ_{ij}. The case from the case library that is most similar to the current case will yield the largest weight. This of course, will be a huge factor in the predicted value of the dependent variable(s). The following are the equations used for Inverse-Distance Weighted Average:

$$\delta_{ij} = \frac{(1/d_{ij})}{\sum_{j \in N}(1/d_{ij})} \tag{6}$$

$$\widehat{y}_i = \sum_{j \in N} \delta_{ij} y_i \tag{7}$$

2.3 Classification Methods

There are several classification methods that can be used to predict the dependent variables in the software quality classification models. Before any classification models can be built, the observations in the fit data set are classified into particular classes. For instance, source files can be classified as fault-prone (FP) or not fault-prone (NFP). The observations could then be classified as FP or NFP based on whether the number of faults exceeds a threshold set by the user. Therefore, the class of an observation can be determined by the following:

$$Class = \begin{cases} FP, & \text{if } y \leq threshold \\ NFP, & \text{otherwise} \end{cases} \tag{8}$$

where, y is the number of faults.

The following are some classification methods that can be used in CBR for classification.

Data Clustering: With the Data Clustering method, the case library is partitioned into clusters or groups according to the actual class of each case. The distances to the clusters are then computed for the current case. The classification of the current case is then determined by comparing the ratio of the average of these distances to the cost ratio. The cost ratio is defined as CI/CII, where CI is the cost of a Type I misclassification, and CII is the cost of a Type II misclassification.

A Type I misclassification is when a classification model classifies a case as fault-prone when it is actually not-fault prone. A Type II misclassification is when a classification model classifies a case as not fault-prone when it is actually fault prone [12]. The Type II misclassifications are generally considered more serious than the Type I misclassifications because they represent the cost of releasing fault-prone cases in the final product. Type I misclassifications, on the other hand, represent the efforts wasted on analyzing low-risk modules.

During model development, the user can experiment with various cost ratios to reach the desired value for their model. Since Type II errors are considered more serious we prefer to select the most balanced Type I and Type II misclassification rates with the Type II misclassification rate being as low as possible.

The classification terminology for Data Clustering is as follows [10]: for an unclassified case, x_i, let $d_{nfp}(x_i)$ be the average distance to the not fault-prone nearest neighbor cases, and let $d_{fp}(x_i)$ be the average distance to the fault-prone nearest neighbor cases. The following is the generalized classification rule for Data Clustering:

$$Class(x_i) = \begin{cases} NFP, & \text{if } \frac{d_{fp}(x_i)}{d_{nfp}(x_i)} > \frac{C_I}{C_{II}} \\ FP, & \text{otherwise} \end{cases} \tag{9}$$

Majority Voting: Majority Voting polls the cases from the nearest neighbor set, N, to determine the classification of the current case. Since a majority consensus is needed, an odd number of cases is required. The probability of classification as either fault-prone versus not fault-prone, for instance, would depend on the percentage of cases within n_N that are fault-prone and not fault-prone. These percentages would represent the probability of the current case being fault-prone and not fault-prone, respectively. The classification of the current case would then depend on whether or not the ratio of these probabilities exceeds a constant c, which is chosen empirically. The following is our generalized classification rule for Majority Voting:

$$Class(x_i) = \begin{cases} NFP, & \text{if } \frac{P_{nfp}}{P_{fp}} > c \\ FP, & \text{otherwise} \end{cases} \tag{10}$$

where, P_{nfp} is the probability that the current (target) case is not fault-prone, and P_{fp} is the probability that the current (target) case is fault-prone.

2.4 Performance Evaluation

The statistics used for performance evaluation depends on the type of model that is being evaluated. For prediction models, the statistics used are the average absolute error, AAE, and the average relative error, ARE. For classification models, the statistics used are the Type I and Type II misclassification rates.

Performance Evaluation for Prediction Models: The following are the equations used for AAE and ARE:

$$AAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{11}$$

$$ARE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i + 1}\right| \tag{12}$$

where 1 is added to the denominator in the computation of ARE to avoid any division by zero.

Performance Evaluation for Classification Models: The misclassification rates are determined by assessing the actual classifications. The model determines the number of cases that are fault-prone and the number of cases that are not fault-prone. The number of misclassified cases is determined and the percentage over the total number of cases is determined to be the misclassification rate. The two misclassification rates are inversely related, such that when experiments are performed using different cost ratios, Type I misclassification rate increase is associated with a corresponding Type II misclassification rate decrease. Inversely, if the Type II misclassification rate increases as the cost ratio changes, then the Type I misclassification rate decreases. Changes in the cost ratio can have a significant effect on the model chosen. Previous research [1], has indicated that cost ratios have an effect on the usefulness of a particular model.

3 Empirical Case Study 1

This section will first describe the system used for our first empirical study. The different metrics collected will be highlighted and then the results will be presented.

3.1 System description

Our research was conducted on two data sets obtained from the industry. The data sets were both collected from the same project within the organization. The project

Table 1: System Profile for Case Studies

Applications	Service Configuration Software
Language	C++
Application 1: AENSCL*	320 million
Application 1: Actual Lines of Code	29 million
Application 2: AENSCL*	300 million
Application 2: Actual Lines of Code	27.5 million
Number of source files	1400

* AENCSL is Assembly Equivalent Non-Commented Source Lines of Code

consisted of two large Windows-based applications used primarily for customizing the configuration of wireless products. The data sets were obtained from the initial release of these applications. The applications are written in C++, and they provide similar functionality. As a result, the applications contain common code. The main difference between the two applications is the type of wireless product that it can support. Table 1 presents the profile of the system used for this case study.

3.2 Data Collection Effort for Case Study 1

For this first case study involving source files, we derived our data set of files by viewing the configuration management system database. A configuration management system database keeps track of changes made to various software development deliverables such as source files [13].

The problem reporting system database was then used to determine the INSP metric used in our case study. A problem reporting system database is used to account for problems logged by tracking the problem from discovery to resolution. The raw data tracked can include problem identifier, problem status, problem severity, date discovered, source files modified, and problem type. The database can also be used to track inspection information as well. The INSP metric is the number of times a source file was inspected prior to system tests. The inspections logged in the problem reporting database enabled us to indirectly determine INSP.

The remaining metrics are related to different measures of lines of code, $BASE\_LOC$, $SYST\_LOC$, $BASE\_COM$, and $SYST\_COM$. They were obtained by using an internally developed tool which has the ability to count the lines of code when source code file versions are provided as input. Table 2 presents the selected product and process metrics.

3.3 Case Study 1 Description

Case Study 1 analyzes our first data set, which contains data on source files. After the data set was preprocessed and cleaned [13], 1211 observations remained. In this particular case, an observation is a source file. One process metric and four product metrics were used as independent variables. The five metrics are described in Table 2. Because this list of raw metrics is smaller compared to other research studies (sometimes more than 20 metrics), it is quite manageable. As a result, data reduction or transformation techniques, such as principal components analysis, (PCA) [3], were not necessary. This set of metrics will be referred to as $RAW\_FILES - 5$. The correlations among the five independent

Table 2: Product and Process Metrics for Case Study 1

Product Metrics	Description
BASE_LOC	Number of lines of code for the source file version just prior to the coding phase. This represents auto-generated code.
SYST_LOC	Number of lines of code for the source file version delivered to system test.
BASE_COM	Number of lines of commented code for the source file version just prior to the coding phase. This represents auto-generated code.
SYST_COM	Number of lines of commented code for the source file version delivered to system test.
Process Metrics	**Description**
INSP	Number of times the source file was inspected prior to system test.

variables are shown in Table 3. Figure 1 displays the frequency of the number of faults in the 1211 source files. The dependent variable in this first case study is the number of faults discovered in a source file during system tests.

Table 3: Correlation Statistics for $RAW\_FILES - 5$

CORRELATION	INSP	BASE_LOC	SYST_LOC	BASE_COM	SYST_COM
INSP	1.00	0.48	0.73	0.16	0.51
BASE_LOC		1.00	0.67	0.51	0.14
SYST_LOC			1.00	0.44	0.69
BASE_COM				1.00	0.21
SYST_COM					1.00

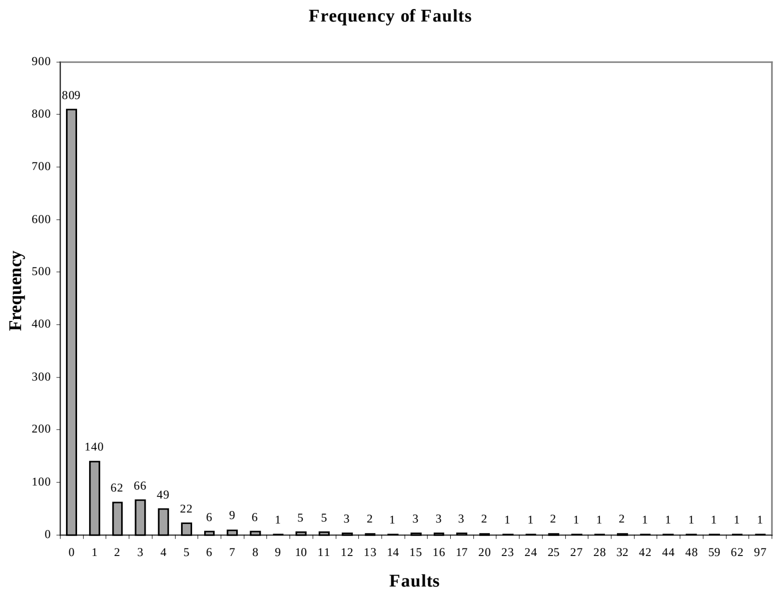

Figure 1: Frequency of faults

3.4 Case Study 1 Methodology

As we already mentioned, user-defined thresholds determine the class of an observation. In our first case study, the fault-proneness of a source file is determined by comparing the number of faults, y, to three possible thresholds.

There are three general steps to software quality modeling. The first step is to build the model. The second step is to validate the model, and the third step is to evaluate its accuracy. There are several techniques available to build and validate software quality models.

Validation techniques can include cross-validation and data splitting. The accuracy of the classification models is evaluated by calculating the Type I and Type II misclassification rates. The accuracy of the prediction models is evaluated by calculating the Average Absolute Error (AAE) and Average Relative Error (ARE).

The following steps describe the modeling methodology used in our first case study for both types of models:

1. Collect configuration management and problem reporting data from a past project.

2. Preprocess and clean the data.

3. Identify the following: the list of independent variables and the dependent variable. We identified five independent variables, and our dependent variable is the number of faults discovered in a source file during system test.

4. For classification modeling: Determine the class of each observation.

$$Class = \begin{cases} FP, & \text{if } y \leq threshold \\ NFP, & \text{otherwise} \end{cases} \tag{13}$$

   where, FP is fault-prone, NFP is not fault-prone, y is the number of faults, and $threshold$ is determined based upon project-specific criteria.

5. Prepare the fit and target data (test data) sets. A fit data set is used to build the model, while a target data set is used to validate the model.

6. Select a modeling methodology and built the model. We used CBR. The models were validated and evaluated for their accuracy. We validated the models using cross-validation and data splitting.

3.5 Classification Experiments

There were six classification experiments conducted in our first case study. The first three experiments involved the entire data set with one experiment conducted for each fault-proneness threshold chosen. The target data (test data) set for these experiments was the same as the fit data set, and the models built were validated using cross-validation. The second set of three experiments involved the use of a fit data set and a separate target data set for building and validating the models. Data from subsequent releases was not available during our research therefore, a fit data set and a target data set were derived from the single data set available. This technique is possible when the data set is large, and it is useful because it provides a simulation of model application in practice. The data set observations were impartially split to create a fit data set and a target data set. The user determines the proportion of observations in each data set. In our research, the fit data set comprised 2/3 of the original data set or 807 observations, and the target data set comprised 1/3 of the original data set or 404 observations. The models built in our second set of classification experiments were validated by cross-validation using the fit data set and by data splitting using the target data set.

Based on the CBR methodology described in section 2, the following are the characteristics of our classification models for Case Study 1:

1. A fit data set is specified as the case library. For the first set of three experiments,

the fit data set is the entire data set. For the second set of experiments, the fit data set is 2/3 of the entire data set.

2. A target data set is specified for model validation. For the first set of experiments, the target data set is the same as the fit data set. For the second set of experiments, the target data set is 1/3 of the entire data set.

3. A similarity function is selected. We selected Mahalonobis Distance.

4. The number of cases is selected. This is a significant parameter that is varied during model selection. The model with the optimum results dictates the optimum value for the number of cases, n_N.

5. For classification models, the classification method is selected. We selected Data Clustering.

6. The models are evaluated for accuracy. We used Type I and Type II misclassification rates.

In addition to the CBR methodology specifications mentioned above, the cost ratio parameter, CI/CII, was also varied to determine the most balanced misclassification rates. The (cost ratio, n_N) parameter combinations were varied for each model built, and the model that yielded the "best results" provided the optimum (cost ratio, n_N) combination. In our research, "best results" is defined as the model that yields the most balanced Type I and Type II misclassification rates with the Type II misclassification rate being as low as possible.

The six classification experiments based on the entire data set and the fit/target data set were conducted for each of the three fault-proneness thresholds. The varying of thresholds in our experiments is a type of sensitivity analysis that allows us to determine whether the models built are sensitive to the thresholds selected by the user. The three fault-proneness thresholds selected for our research are the following:

- *Threshold 1*: a file is fault-prone if it contains one or more faults: $y \geq 1$
- *Threshold 2*: a file is fault-prone if it contains two or more faults: $y \geq 2$
- *Threshold 3*: a file is fault-prone if it contains three or more faults: $y \geq 3$

The main stimulus for selecting appropriate thresholds is to be able to build the most useful and relevant quality model possible. If a threshold is set too low, the model may classify all observations as fault-prone. Similarly, if a threshold is set too high, the model may classify all observations as not fault-prone. In both situations, the models would not reveal any useful information. Therefore, it is important to set a reasonable fault-proneness threshold for the particular data available in order to build the optimum software quality model.

Classification Experiment 1- Entire Data Set, Threshold 1: When *Threshold 1* is used, 402 source files or 33% of the 1211 source files are deemed to be FP. The remaining 809 source files or 67% are deemed to be NFP. Data clustering was used with CBR to classify the files. Each observation (source file) within the fit data set was analyzed and placed into its appropriate FP or NFP cluster, based upon the threshold classification stated above. Using cross-validation, we built and validated the model. The Mahalonobis Distance similarity function was used to determine the distance from the FP nearest neighbor cases and from the NFP nearest neighbor cases. Based on previous research [21], the Mahalonobis Distance reveals a high performance accuracy

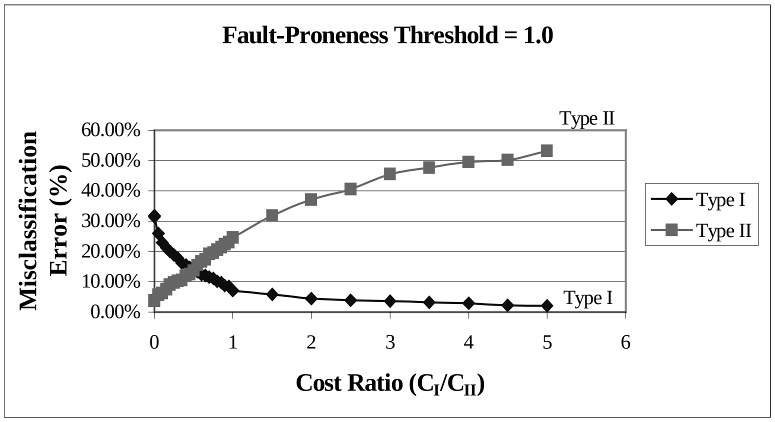

Figure 2: Type I and Type II Misclassification Rates Per Cost Ratio- Threshold 1

for raw data sets. The ratio between the average of these distances was compared to the cost ratio. This comparison to the cost ratio dictates to which cluster, the current observation is to be classified.

In our research, we varied the cost ratio and the number of cases representing the nearest neighbors. First, for each n_N scenario, we varied the cost ratio. The cost ratio that provided the most balanced Type I and Type II misclassification rates was determined to be the optimum cost ratio for that particular n_N. Table 4 identifies the cost ratios and their associated Type I and Type II misclassification rates where $n_N = 1$. Figure 2 displays the tradeoff between the misclassification rates as a function of the cost ratio. This demonstrates the inverse relationship between the Type I and Type II misclassification rates. Once the optimum cost ratio for each n_N was determined, the results were tabulated and the (cost ratio, n_N) combination with the most balanced misclassification rates was selected to be the overall optimum model. Table 5 displays the model results for Experiment 1.

Table 4 indicates that for the models where $n_N = 1$, the optimum cost ratio is 0.5

because it provided the most balanced misclassification rates. Figure 2 shows the inverse relationship between the misclassification rates. As the cost ratio increases, the Type II misclassification rate also increases. Since Type II errors are generally more costly than Type I errors, models with cost ratios greater than 1 would more than likely yield inappropriate models. Table 5 indicates that for *Threshold 1*, the optimum model is where $n_N = 1$ and the cost ratio $= 0.5$.

It is interesting to note how a model with $n_N = 1$ yielded the best results. Within the source file data set, it was determined that there were several groups of source files that contained the same exact values for all five of the independent variables selected. This indicates that when an observation is selected for comparison to the observations in the case library, an exact match can be found. As a result, the classification is perfect. This perfect classification is unique to CBR which reveals why it is an attractive methodology to implement. When an exact match is found, no errors are calculated because the classification is perfect. This explains the excellent performance that can be obtained when using CBR.

For *Threshold 1*, the optimum model shows a Type I misclassification rate of 14.34% and a Type II misclassification rate of 13.68%. Following the same analysis posed in [14], suppose we propose to increase our reliability by conducting extra reviews for those source files that are suspected of being FP at a cost of one unit. If any of the files were later determined to be NFP, then the extra reviews would simply be a waste of time. In our experiment, the best cost ratio was determined to be 0.50. Therefore, if the cost of a Type I misclassification is one unit, then the cost of a Type II misclassification is two (1/0.50) units. However, the two units actually represent the net cost of forfeited benefit. Because the cost of extra reviews is one unit, the actual cost of a Type II misclassification is three units. The three units consist of the cost of the forfeited benefit plus the cost of reviews. A perfect model for *Threshold 1* would cost 402 units for review and yield

$$402 * 3 = 1206 \tag{14}$$

units in avoided debugging costs. When this model is compared to the optimum model for Experiment 1, the expected cost of misclassifications would be

$$(809 * 0.1434 * 1) + (402 * 0.1368 * (3 - 1)) = 226 \tag{15}$$

units. The cost of reviews would be

$$(809 * 0.1434 * 1) + (402 * 0.8632 * 1) = 463 \tag{16}$$

units. The benefit would be reliability improvement for

$$(402 * 0.8632) = 347 \tag{17}$$

fault-prone files. This might avoid

$$(347 * 3) = 1041 \tag{18}$$

units in debugging costs later. This would be considered a good return on investment (ROI), 1041:463. The model identifies the FP files that need extra reviews. In this experiment, the cost of the extra reviews for the FP files would be 463 units. By focusing efforts on the source files that are deemed to be FP early, managers could potentially save a net of

$$(347 * 2) = 694 \tag{19}$$

units of debugging costs in the future.

Table 4: Cost Ratio Misclassification where n_N=1-Threshold 1

	Fit(Cross - Validation)	
Cost ratio CI/CII	Type I	Type II
0.0005	31.77%	3.73%
0.001	31.77%	3.73%
0.0015	31.64%	3.73%
0.002	31.64%	3.73%
0.0025	31.40%	3.73%
0.05	25.96%	5.72%
0.1	22.99%	6.22%
0.15	21.26%	7.46%
0.2	20.03%	8.96%
0.25	18.91%	9.70%
0.3	17.80%	10.20%
0.35	16.19%	10.45%
0.4	15.58%	11.94%
0.45	14.83%	12.44%
0.5	**14.34%**	**13.68%**
0.55	13.23%	15.42%
0.6	12.36%	16.67%
0.65	11.99%	17.41%
0.7	11.50%	19.15%
0.75	11.13%	19.65%
0.8	10.14%	20.65%
0.85	9.64%	21.39%
0.9	8.78%	22.39%
0.95	8.41%	23.13%
1	7.05%	24.63%
1.5	5.81%	31.84%
2	4.45%	37.07%
2.5	3.83%	40.55%
3	3.59%	45.52%
3.5	3.21%	47.51%
4	2.97%	49.50%
4.5	2.23%	50.25%
5	2.10%	53.23%

Table 5: Classification Model Results-Entire Data Set-Threshold 1

n_N	CI/CII	Fit(Cross - Validation)	
1	**0.5**	**14.34%**	**13.68%**
2	0.55	15.33%	14.43%
3	0.5	15.95%	15.17%
4	0.55	16.19%	15.92%
5	0.6	16.07%	15.92%
7	0.6	16.32%	16.92%
9	0.6	17.06%	18.66%
11	0.6	18.17%	17.91%
13	0.55	18.91%	20.15%
15	0.55	18.17%	19.40%
17	0.55	18.67%	19.15%
19	0.5	20.15%	19.65%
21	0.5	19.41%	19.90%
23	0.5	19.41%	20.15%
25	0.5	19.90%	20.40%
27	0.5	20.40%	19.90%
29	0.5	21.88%	19.90%
31	0.55	21.01%	22.39%
33	0.55	23.12%	22.14%
35	0.6	21.88%	23.88%
37	0.6	21.76%	24.38%
39	0.6	22.87%	22.14%
41	0.6	23.24%	25.62%
43	0.55	27.32%	23.38%
45	0.55	27.44%	25.62%
47	0.55	27.19%	27.86%
49	0.55	27.19%	28.61%
51	0.55	27.19%	28.61%
53	0.55	27.69%	29.10%
55	0.55	28.43%	31.59%
57	0.55	28.93%	31.59%
59	0.55	29.42%	31.59%
61	0.55	30.04%	31.59%
63	0.55	30.04%	31.59%
65	0.55	30.16%	31.59%

Table 6: Evaluation of Entire Data Set-Threshold 1

Model (Cross-Validation)			
Actual	NFP	FP	TOTAL
NFP	693 85.66%	116 14.34%	809 100%
FP	55 13.68%	347 86.32%	402 100%
TOTAL	748	463	1211
PERCENT	61.77%	38.23%	100%
PRIOR	66.80%	33.20%	

Table 6 provides the evaluation results for the optimum model. Using this model, 347/402 of the FP source files are correctly classified. Similarly, 693/809 of the NFP source files are correctly classified. From this model, we would expect to classify 463 files as being FP. This represents the sum of the files that were correctly classified as FP, and the files that were misclassified as FP. If extra effort is invested in improvimg the reliability of these 463 files, then the fraction of the FP files could be reduced from 402/1211 or 33.20% to 4.54% (0.3320 * 0.1368).

Classification Experiment 2- Entire Data Set, Threshold 2: When *Threshold 2* is used, 262 files or 22% of the 1211 files are deemed to be FP. The remaining 949 files or 78% are deemed to be NFP. The same methodology used in Experiment 1 was followed. Table 7 identifies the cost ratios and their associated Type I and Type II misclassification

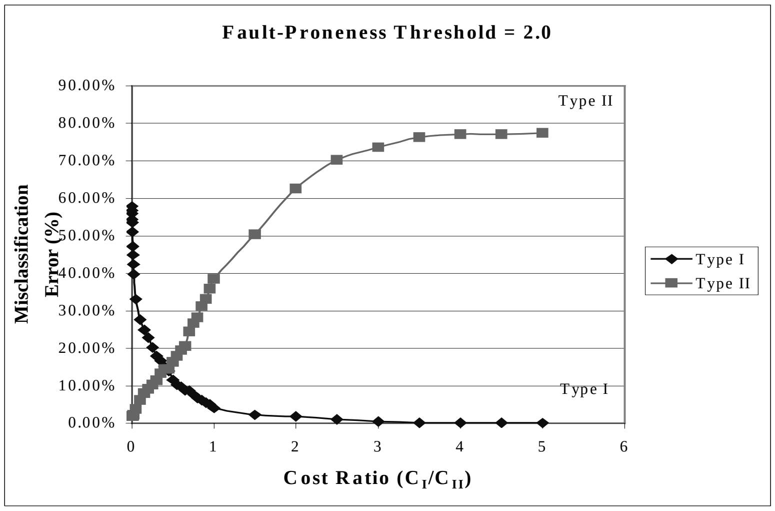

Figure 3: Type I and Type II Misclassification Rates Per Cost Ratio- Threshold 2

rates where $n_N = 4$. Figure 3 displays the tradeoff between the misclassification rates as a function of the cost ratio. Table 7 displays the model results for Experiment 2.

Table 7 indicates that for $n_N = 4$, the optimum cost ratio is 0.45. Figure 3 shows the inverse relationship between the misclassification rates. Table 7 indicates that for *Threshold 2*, the optimum model is $n_N = 4$ and the cost ratio = 0.45.

The optimum model for *Threshold 2*, shows a Type I misclassification rate of 13.91% and a Type II misclassification rate of 14.50%. Following the same analysis for Experiment 1, and given a cost ratio of 0.45, a Type I misclassification cost would be one unit, and a Type II misclassification cost would be 2.22 (1/0.45) units. Therefore, the actual cost of a Type II misclassification would be 3.22, which is 2.22 plus the one unit cost of extra reviews. A perfect model for *Threshold 2* would cost 262 units for review and yield 844 (262*3.22) units in avoided debugging costs. When this model is compared to the optimum model for Experiment 2, the expected cost of misclassifications would be 216 (949*0.1391*1)+(262*0.1450*(3.22-1)) units. The cost of reviews would be 356

Table 7: Cost Ratio Misclassification where n_N=4-Threshold 2

	Fit(Cross - Validation)	
Cost ratio CI/CII	Type I	Type II
0.0005	57.85%	1.91%
0.001	56.80%	1.91%
0.0015	55.95%	1.91%
0.002	54.37%	1.91%
0.0025	53.53%	1.91%
0.005	51.11%	1.91%
0.01	47.21%	1.91%
0.015	45.00%	1.91%
0.02	42.47%	2.29%
0.025	39.73%	2.29%
0.05	33.09%	3.82%
0.1	27.61%	6.11%
0.15	24.97%	8.02%
0.2	22.87%	9.16%
0.25	20.23%	10.31%
0.3	17.91%	11.45%
0.35	16.75%	13.36%
0.4	15.17%	14.50%
0.45	**13.91%**	**14.50%**
0.5	11.59%	16.41%
0.55	10.33%	17.94%
0.6	9.69%	19.47%
0.65	8.85%	20.61%
0.7	8.75%	24.43%
0.75	7.59%	26.72%
0.8	6.74%	28.24%
0.85	6.11%	31.30%
0.9	5.59%	33.21%
0.95	5.06%	35.88%
1	4.11%	38.55%
1.5	2.21%	50.38%
2	1.79%	62.60%
2.5	1.05%	70.23%
3	0.53%	73.66%
3.5	0.21%	76.34%
4	0.21%	77.10%
4.5	0.21%	77.10%
5	0.11%	77.48%

Table 7: Classification Model Results-Entire Data Set-Threshold 2

n_N	CI/CII	Fit(Cross - Validation)	
1	0.35	14.23%	15.27%
2	0.4	14.86%	15.65%
3	0.4	14.86%	14.89%
4	**0.45**	**13.91%**	**14.50%**
5	0.45	14.02%	15.27%
6	0.45	14.75%	15.65%
7	0.4	16.33%	14.89%
8	0.45	15.17%	16.79%
9	0.45	15.39%	16.41%
10	0.45	15.39%	16.41%
11	0.5	15.17%	16.03%
13	0.5	15.81%	16.79%
15	0.5	16.33%	16.79%
17	0.5	16.23%	16.79%
19	0.5	16.33%	16.79%
21	0.45	17.60%	16.03%
23	0.5	16.12%	17.56%
25	0.35	21.29%	20.23%
27	0.35	21.29%	21.76%
29	0.3	26.34%	21.76%
31	0.3	25.82%	23.28%
33	0.3	25.92%	23.28%
35	0.3	26.87%	24.05%
37	0.3	28.03%	27.48%
39	0.3	29.51%	27.10%
41	0.35	25.18%	30.53%
43	0.35	25.50%	28.63%
45	0.35	26.34%	28.24%
47	0.35	29.82%	27.48%
49	0.35	30.88%	27.48%
51	0.3	31.51%	27.10%
53	0.35	32.67%	28.24%
55	0.35	32.88%	27.86%
57	0.35	33.51%	28.24%
59	0.35	33.83%	27.48%
61	0.35	34.98%	27.48%
63	0.4	23.29%	31.30%
65	0.4	23.71%	31.30%

Table 8: Evaluation of Entire Data Set-Threshold 2

Model (Cross-Validation)			
Actual	NFP	FP	TOTAL
NFP	817 86.09%	132 13.91%	949 100%
FP	38 14.50%	224 85.50%	262 100%
TOTAL	855	356	1211
PERCENT	70.60%	29.40%	100%
PRIOR	78.36%	21.64%	

(949*0.1391*1) + (262*0.8550*1) units. The benefit would be reliability improvement for 224 (262*0.8550) fault-prone files, which might avoid 721 (224*3.22) units in debugging costs later. The *ROI* is 721:356.

Table 8 provides the evaluation results for the optimum model. Using this model, 224/262 of the FP files are correctly classified. Similarly, 817/949 of the NFP files are correctly classified. From this model, we would expect to classify 356 files as being FP. If extra effort is invested to improve the reliability of these 356 files, then the fraction of the FP files could be reduced from 262/1211 or 21.64% to 3.13% (0.2164 * 0.1450).

Classification Experiment 3- Entire Data Set, Threshold 3: When *Threshold 3* is used, 200 files or 17% of the 1211 files are deemed to be FP. The remaining 1011 files or 83% are deemed to be NFP. The same methodology used in Experiment 1 was followed.

Table 9 identifies the cost ratios and their associated Type I and Type II misclassification rates where $n_N = 6$. Figure 4 displays the tradeoff between the misclassification rates as a function of the cost ratio. Table 9 displays the results for Experiment 3.

Table 9 indicates that for $n_N = 6$, the optimum cost ratio is 0.40. Table 9 indicates that for Threshold 3, the optimum model is where $n_N = 6$ and the cost ratio equal to 0.40.

The optimum model for *Threshold 3*, shows a Type I misclassification rate of 14.74% and a Type II misclassification rate of 14.00%. Following the same analysis for Experiment 1, and given a cost ratio of 0.40, a Type I misclassification cost would be one unit, and a Type II misclassification cost would be 2.50 (1/0.40) units. Therefore, the actual cost of a Type II misclassification would be 3.50, which is 2.50 plus the one unit cost of extra reviews. A perfect model for *Threshold 3* would cost 200 units for review and yield 700 (200*3.5) units in avoided debugging costs. When this model is compared to the optimum model for Experiment 3, the expected cost of misclassifications would be 219 (1011*0.1474*1)+(200*0.1400*(3.50-1)) units. The cost of reviews would be 321 (1011*0.1474*1) + (200*0.8600*1) units. The benefit would be reliability improvement for 172 (200*0.8600) FP files, which might avoid 602 (172*3.50) units in debugging costs later. The ROI is 602:321. Table 10 provides the evaluation results for the optimum model. Using this model, 172/200 of the FP source files are correctly classified. Similarly, 862/1011 of the NFP source files are correctly classified. From this model, we would expect to classify 321 files as being FP. If extra effort is then invested to improve the reliability of these 321 files, then the fraction of the FP files could be reduced from 172/1211 or 16.52% to 2.31% (0.1652 * 0.1400).

Table 9: Cost Ratio Misclassification where n_N=6-Threshold 3

	Fit(Cross - Validation)	
Cost ratio CI/CII	Type I	Type II
0.0005	64.39%	0.50%
0.001	62.91%	0.50%
0.0015	61.52%	0.50%
0.002	59.94%	0.50%
0.0025	59.25%	0.50%
0.005	56.38%	0.50%
0.01	52.92%	0.50%
0.015	50.45%	1.00%
0.02	46.29%	1.00%
0.025	42.93%	1.50%
0.05	35.61%	2.00%
0.1	29.87%	4.00%
0.15	28.29%	5.50%
0.2	25.72%	7.50%
0.25	22.75%	9.00%
0.3	20.08%	11.00%
0.35	17.01%	12.50%
0.4	**14.74%**	**14.00%**
0.45	13.16%	15.00%
0.5	12.17%	17.50%
0.55	10.39%	20.50%
0.6	9.00%	24.00%
0.65	7.52%	26.50%
0.7	6.53%	31.50%
0.75	5.84%	33.00%
0.8	5.24%	34.50%
0.85	4.85%	36.50%
0.9	4.55%	38.00%
0.95	4.06%	40.50%
1	3.56%	41.50%
1.5	2.18%	55.50%
2	1.78%	72.00%
2.5	1.48%	75.50%
3	1.19%	78.00%
3.5	1.19%	80.00%
4	1.19%	81.00%
4.5	1.09%	82.00%
5	1.09%	82.50%

Table 9: Classification Model Results-Entire Data Set-Threshold 3

n_N	CI/CII	Fit(Cross - Validation)	
1	0.25	15.93%	16.50%
2	0.3	15.63%	14.50%
3	0.3	16.72%	13.50%
4	0.3	17.51%	13.50%
5	0.35	16.72%	15.50%
6	**0.4**	**14.74%**	**14.00%**
7	0.4	15.33%	13.50%
8	0.4	16.12%	13.50%
9	0.45	14.44%	16.00%
10	0.4	17.01%	13.00%
11	0.4	17.71%	15.00%
13	0.4	19.19%	14.00%
15	0.4	19.09%	16.00%
17	0.35	20.67%	19.50%
19	0.35	20.48%	19.50%
21	0.35	20.67%	19.50%
23	0.25	25.12%	22.50%
25	0.25	26.31%	24.50%
27	0.25	26.81%	25.00%
29	0.25	27.30%	25.00%
31	0.25	27.50%	25.00%
33	0.25	27.99%	25.00%
35	0.25	28.39%	25.00%
37	0.25	29.38%	27.00%
39	0.25	29.67%	27.00%
41	0.25	29.77%	27.00%
43	0.25	29.77%	27.00%
45	0.25	30.17%	26.50%
47	0.25	30.47%	26.50%
49	0.25	30.47%	26.00%
51	0.25	30.47%	26.00%
53	0.25	29.77%	26.00%
55	0.25	29.87%	27.50%
57	0.25	29.97%	27.50%
59	0.25	30.07%	27.50%
61	0.25	30.17%	27.50%
63	0.25	29.87%	27.50%
65	0.25	29.87%	27.50%

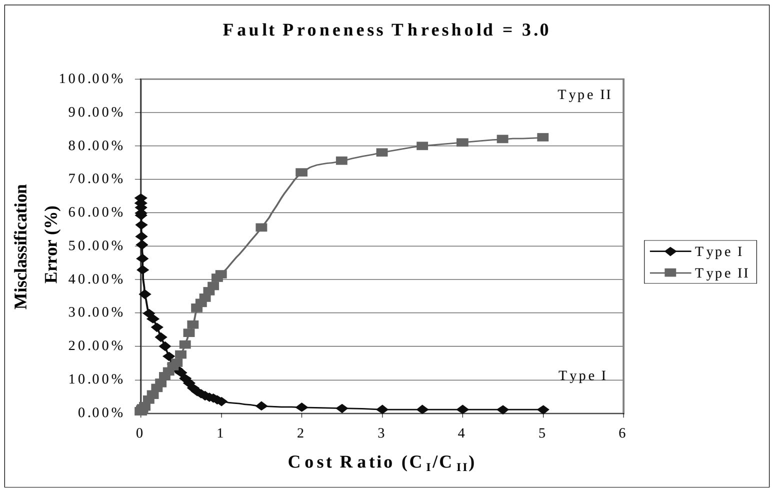

Figure 4: Type I and Type II Misclassification Rates Per Cost Ratio- Threshold 3

Table 10: Evaluation of Entire Data Set-Threshold 3

Model (Cross-Validation)			
Actual	NFP	FP	TOTAL
NFP	862 85.26%	149 14.74%	1011 100%
FP	28 14.00%	172 86.00%	200 100%
TOTAL	890	321	1211
PERCENT	73.49%	26.51%	100%
PRIOR	83.48%	16.52%	

Table 11: Classification Model Results-Entire Data Set Experiments

Threshold	Type I	Type 2	ROI	Prior (%)	Reduction (%)
			Debug cost	Fault-prone	Fault-prone
1	14.34%	13.68%	1041:463	33.20%	4.54%
2	13.91%	14.50%	721:356	21.64%	3.13%
3	14.74%	14.00%	602:321	16.52%	2.13%

Classification Experiments- Entire Data Set Results: Table 11 displays the results for the three classification experiments conducted when the entire data set was used to build the model, and cross-validation was used to validate the model. For each of the thresholds, we obtained similar Type I and Type II misclassification rates. In addition, the ROI for each of the thresholds were also quite similar. If managers were to implement any of the three classification models, the percentage of FP source files could be reduced to less than 5%. In addition, they could expect to save anywhere from 47% ((602-321)/602) to 56% ((1041-460)/1041) in debugging costs. However, these ROI figures are only based on our modeling analysis. In reality, the actual ROI may be much greater. This is attributed to the fact that a higher quality product could be delivered to customers. Customer satisfaction would be greater which could help a company's market share and image, and the cost of fixing faults late in the project would be minimized. In addition, the actual cost of latent faults discovered by customers may be higher than the costs obtained by our CBR modeling. These are just some of the benefits that allow the ROI for building software quality models to be much greater.

Classification Experiments 4- Fit/Target Data Set, Threshold 1: The second set of classification experiments involved the use of a fit data set and a target data (test data) set. Because the data set is split impartially, the usefulness of a model can be impacted by how the data happened to be split. Therefore, in order to have confidence in the model results, several models should be built using different data splits. For our classification experiments, we impartially split the data set 50 times and built 50 different classification models. All 50 models were validated using cross-validation and data splitting. The misclassification rate results for *Threshold 1* using both model validation techniques are tabulated in table 12, and the average misclassification rates and their standard deviations are provided as well. The same two parameters, cost ratio and n_N , were varied for each model. The cross-validation results were used to determine the cost ratio and n_N parameters for the models that were validated using data splitting.

The average Type I and Type II rates for the models validated using cross-validation are 15.91% and 15.55%, respectively. The average Type I and Type II rates for the models validated using data splitting are 15.75% and 16.37%, respectively. The average n_N value is 2.28, and the average cost ratio is 0.57. The results show values ranging from 1 to 9. This indicates that this parameter should not be fixed at a predetermined value. Instead, n_N should be chosen empirically for each case. The cost ratio parameter is more uniform than the n_N value, which is evident by the small standard deviation.

Classification Experiments 5- Fit/Target Data Set, Threshold 2: Table 12 displays the results for all 50 models for *Threshold 2*. The average Type I and Type II misclassification rates for the models validated using cross-validation are 15.09% and 14.71%, respectively. The average Type I and Type II misclassification rates for the models validated using data splitting are 15.08% and 16.19%, respectively. The average n_N value is 2.10, and the average cost ratio is 0.40. The results show n_N values ranging from 1 to 7, which is similar to Classification Experiment 4, indicates that the parameter should be chosen empirically to obtain the optimum model for each case.

Classification Experiments 6- Fit/Target Data Set, Threshold 3: Table 12 displays the results for all 50 models for *Threshold 3*. The average Type I and Type II misclassification rates for the models validated using cross-validation are 15.65% and 15.28%, respectively. The average Type I and Type II misclassification rates for the models validated using data splitting are 15.08% and 16.55%, respectively. The average n_N value is 3.16, and the average cost ratio is 0.37. For *Threshold 3*, the n_N values range from 1 to 5. The standard deviation for the cost ratio parameter is 0.054.

Classification Experiments - All Experiments Results: Table 12 summarizes the results from the six classification experiments conducted. Splitting the data set into a fit data set and a target data set further validated the models built from the independent variables selected in our study. Data splitting offers the ability to simulate the use of the model in practice because the target data set can represent data from subsequent releases. 50 models were built to ensure that the results obtained were due to accurate modeling instead of a fortunate data split. The data splitting results were quite similar to the cross-validation results which provides additional confidence that successful classification models were built. The Type II errors were a bit higher for the models validated using data splitting, but the difference is not significant. The cross-validation results from the entire data set was also quite similar to the cross-validation results which used 2/3 of

Table 12: Classification Model Results-Fit/Target Data Set-Threshold 1

Data Set	n_N	CI/CII	Fit(Cross Validation)		Target(Test)	
			Type I	Type II	Type I	Type II
1	4	0.6	16.14%	16.79%	14.44%	13.43%
2	1	0.5	14.47%	14.93%	18.52%	15.67%
3	1	0.6	16.51%	16.79%	17.41%	14.93%
4	1	0.6	16.51%	16.79%	17.41%	14.93%
5	2	0.6	13.57%	13.38%	14.39%	22.56%
6	6	0.6	16.33%	16.79%	18.52%	20.15%
7	3	0.5	15.77%	14.18%	14.44%	14.18%
8	9	0.6	17.22%	16.11%	17.47%	18.52%
9	2	0.6	15.58%	13.81%	18.89%	14.93%
10	2	0.55	14.60%	15.04%	15.30%	18.38%
11	3	0.6	15.86%	15.87%	19.41%	12.98%
12	5	0.6	16.24%	15.85%	15.36%	21.17%
13	1	0.55	17.07%	16.05%	9.63%	13.43%
14	1	0.55	15.80%	15.24%	17.71%	18.05%
15	2	0.6	15.86%	16.24%	16.48%	14.50%
16	2	0.55	16.27%	16.54%	12.31%	12.50%
17	1	0.55	16.36%	16.35%	14.34%	20.14%
18	1	0.55	17.25%	16.05%	11.48%	14.18%
19	2	0.55	14.31%	14.13%	18.08%	18.05%
20	1	0.55	16.79%	15.87%	16.12%	18.32%
21	1	0.55	16.57%	15.56%	13.97%	14.39%
22	3	0.6	16.57%	15.56%	11.11%	20.98%
23	1	0.5	15.71%	16.17%	15.67%	11.77%
24	1	0.55	15.46%	14.07%	15.44%	21.97%
25	1	0.6	17.07%	17.15%	13.41%	13.28%
26	2	0.5	13.92%	13.41%	16.73%	15.60%
27	5	0.65	16.30%	15.33%	12.93%	24.82%
28	2	0.55	15.77%	14.93%	15.56%	13.43%
29	3	0.55	16.36%	14.13%	20.30%	12.78%
30	1	0.5	16.24%	15.47%	17.23%	14.60%
31	1	0.5	16.24%	15.47%	21.72%	18.25%
32	2	0.6	14.60%	15.04%	12.31%	22.06%
33	2	0.55	16.61%	15.47%	16.61%	15.47%
34	2	0.6	15.95%	14.60%	12.68%	19.53%
35	1	0.55	16.24%	16.60%	13.70%	12.69%
36	1	0.55	14.63%	15.73%	13.38%	16.30%
37	4	0.55	15.75%	15.71%	20.91%	19.15%
38	2	0.6	15.95%	15.69%	13.77%	14.84%
39	1	0.6	16.57%	17.41%	18.38%	13.64%
40	3	0.55	17.64%	16.79%	19.12%	12.88%
41	3	0.6	18.11%	16.25%	14.34%	11.20%
42	5	0.55	16.06%	16.22%	18.39%	18.88%
43	2	0.55	15.57%	14.96%	15.22%	17.97%
44	1	0.5	15.31%	15.09%	14.23%	15.33%
45	5	0.6	14.95%	15.09%	17.23%	16.79%
46	2	0.6	15.61%	15.24%	15.87%	19.55%
47	2	0.6	16.36%	15.59%	13.21%	17.27%
48	2	0.6	15.95%	15.69%	13.77%	14.84%
49	2	0.55	14.89%	15.21%	17.33%	10.24%
50	1	0.55	14.15%	15.21%	15.09%	17.27%
Average	2.28	0.57	15.91%	15.55%	15.75%	16.37%
STD	1.629	0.037	0.95%	0.94%	2.62%	3.30%

Table 12: Classification Model Results-Fit/Target Data Set-Threshold 2

Data Set	n_N	CI/CII	Fit(Cross Validation)		Target(Test)	
			Type I	Type II	Type I	Type II
1	1	0.4	15.19%	14.86%	15.77%	13.79%
2	2	0.4	13.74%	14.37%	16.46%	22.73%
3	3	0.45	12.95%	12.64%	13.29%	19.32%
4	2	0.45	13.90%	14.94%	13.29%	20.46%
5	2	0.35	15.82%	16.00%	14.51%	14.94%
6	2	0.45	13.63%	14.21%	16.04%	16.28%
7	1	0.35	15.48%	15.52%	14.56%	14.77%
8	2	0.4	14.40%	14.29%	13.57%	19.46%
9	1	0.4	14.53%	14.94%	14.87%	11.36%
10	1	0.4	14.29%	14.12%	17.24%	16.47%
11	3	0.4	16.43%	14.94%	15.19%	13.64%
12	1	0.35	14.49%	13.37%	14.65%	13.33%
13	7	0.45	17.35%	16.19%	14.92%	13.48%
14	1	0.4	14.01%	13.97%	18.38%	15.66%
15	1	0.35	15.37%	14.77%	15.72%	16.28%
16	1	0.35	15.64%	15.52%	16.46%	18.18%
17	1	0.4	14.69%	12.64%	14.87%	18.18%
18	3	0.4	17.09%	16.57%	12.93%	14.94%
19	2	0.45	11.57%	10.80%	16.35%	24.42%
20	2	0.35	16.35%	14.62%	16.61%	15.39%
21	3	0.45	13.97%	15.29%	15.39%	13.04%
22	1	0.3	14.96%	15.12%	16.88%	13.33%
23	2	0.4	13.61%	14.29%	14.83%	19.54%
24	2	0.4	14.44%	14.77%	16.04%	18.61%
25	3	0.4	15.24%	13.56%	12.23%	17.65%
26	2	0.4	13.52%	13.45%	16.29%	23.08%
27	2	0.4	15.13%	14.46%	11.04%	18.75%
28	3	0.4	16.93%	16.57%	14.51%	9.20%
29	4	0.45	15.56%	15.82%	16.30%	15.29%
30	1	0.3	16.77%	15.39%	18.97%	7.53%
31	3	0.45	14.83%	14.44%	13.98%	15.85%
32	2	0.4	15.14%	16.19%	13.02%	22.47%
33	2	0.4	13.90%	14.94%	18.04%	14.77%
34	1	0.4	16.48%	17.05%	12.89%	17.44%
35	3	0.4	16.09%	15.03%	11.43%	11.24%
36	1	0.4	15.24%	14.69%	13.79%	15.29%
37	1	0.35	14.38%	13.77%	15.21%	18.95%
38	2	0.4	15.31%	14.44%	14.91%	14.63%
39	1	0.35	15.32%	15.52%	16.46%	14.77%
40	1	0.4	15.39%	13.53%	13.78%	18.48%
41	2	0.35	16.19%	14.12%	14.42%	12.94%
42	6	0.45	14.51%	14.45%	17.46%	21.35%
43	2	0.4	16.22%	14.61%	15.31%	11.91%
44	1	0.4	13.92%	14.29%	15.14%	20.69%
45	2	0.4	16.19%	16.95%	14.42%	14.12%
46	3	0.4	15.53%	14.47%	14.47%	19.77%
47	1	0.4	15.19%	15.43%	14.20%	14.94%
48	2	0.35	14.51%	14.45%	16.83%	12.36%
49	3	0.4	16.69%	15.70%	15.61%	12.22%
50	4	0.4	16.30%	13.50%	14.57%	16.16%
Average	2.1	0.4	15.09%	14.71%	15.08%	16.19%
STD	1.249	0.037	1.17%	1.15%	1.66%	3.62%

Table 12: Classification Model Results-Fit/Target Data Set-Threshold 3

Data Set	n_N	CI/CII	Fit(Cross Validation)		Target(Test)	
			Type I	Type II	Type I	Type II
1	2	0.3	15.13%	14.29%	15.13%	14.93%
2	3	0.4	14.86%	15.67%	18.34%	19.70%
3	3	0.35	14.96%	14.39%	13.99%	16.18%
4	2	0.35	14.67%	15.91%	13.39%	20.59%
5	4	0.45	14.71%	13.43%	13.02%	24.24%
6	5	0.4	16.02%	17.29%	14.54%	13.43%
7	3	0.35	16.02%	14.29%	14.84%	19.40%
8	3	0.35	16.62%	14.29%	14.84%	17.91%
9	3	0.35	16.00%	15.91%	15.77%	10.29%
10	2	0.35	12.84%	13.87%	16.13%	17.46%
11	3	0.35	17.75%	17.56%	15.22%	14.49%
12	3	0.45	11.72%	12.03%	14.84%	22.39%
13	3	0.35	15.90%	17.91%	13.91%	9.09%
14	3	0.4	14.78%	16.06%	16.42%	12.70%
15	2	0.3	15.24%	15.27%	15.82%	17.39%
16	1	0.25	17.06%	16.54%	17.22%	17.81%
17	3	0.4	14.18%	13.87%	15.25%	19.05%
18	5	0.45	14.24%	14.29%	12.46%	20.90%
19	5	0.4	16.87%	13.87%	13.78%	12.70%
20	1	0.3	14.92%	16.15%	15.27%	20.00%
21	2	0.35	17.67%	16.41%	15.36%	11.11%
22	3	0.4	15.88%	14.29%	15.43%	11.94%
23	3	0.35	16.79%	14.93%	17.46%	18.18%
24	4	0.4	14.52%	14.39%	16.96%	16.18%
25	4	0.4	14.67%	15.91%	13.69%	20.59%
26	4	0.4	15.73%	13.53%	15.73%	20.90%
27	1	0.3	15.17%	14.84%	13.25%	20.83%
28	5	0.4	18.69%	17.29%	16.02%	7.46%
29	4	0.4	17.26%	16.30%	15.63%	9.23%
30	5	0.4	15.42%	14.29%	14.24%	17.57%
31	3	0.4	15.44%	14.29%	15.12%	11.67%
32	3	0.35	15.13%	15.79%	14.54%	19.40%
33	1	0.25	15.73%	16.54%	18.99%	13.43%
34	5	0.45	15.90%	15.67%	10.65%	19.70%
35	2	0.3	17.16%	16.03%	13.73%	14.49%
36	4	0.45	14.44%	14.82%	11.21%	16.29%
37	2	0.35	15.68%	13.74%	15.82%	18.84%
38	5	0.45	15.74%	15.71%	14.54%	18.33%
39	3	0.3	17.83%	15.67%	18.94%	9.09%
40	4	0.45	16.02%	15.04%	12.76%	13.43%
41	3	0.35	17.39%	18.66%	14.79%	21.21%
42	2	0.35	13.08%	14.93%	15.39%	25.76%
43	4	0.4	16.62%	14.29%	15.73%	14.93%
44	2	0.35	13.76%	14.50%	14.03%	15.94%
45	4	0.45	14.88%	15.56%	12.98%	16.92%
46	4	0.35	18.48%	16.91%	15.88%	14.06%
47	3	0.4	13.13%	15.33%	15.25%	11.11%
48	1	0.25	17.66%	16.54%	20.48%	17.91%
49	4	0.4	15.46%	14.84%	13.55%	22.22%
50	5	0.4	16.64%	14.06%	15.66%	18.06%
Average	3.16	0.37	15.65%	15.28%	15.08%	16.55%
STD	1.201	0.054	1.46%	1.30%	1.84%	4.21%

Table 12: Classification Model Results-All Experiments

T	Cross-Validation Entire Data Set		Cross-Validation Fit/Target		Data Splitting Fit/Target		Average	
	Type I	Type II	Type I	Type II	Type I	Type II	n_N	CI/CII
1	14.34%	13.68%	15.91%	15.55%	15.75%	16.37%	2.28	0.57
2	13.91%	14.50%	15.09%	14.71%	15.08%	16.19%	2.10	0.40
3	14.74%	14.00%	15.65%	15.28%	15.08%	16.55%	3.16	0.37

the data set as the fit data set. In addition, the results were similar for the three fault-proneness thresholds. This indicates that our classification models were not sensitive to the particular fault-proneness thresholds chosen in our research.

3.6 Prediction Experiments

There were two prediction experiments conducted in our first case study. The first experiment involved the entire data set and the models were validated using cross-validation. The second experiment involved the use of a fit data set and a separate target data set for building and validating the models. Once again, data from subsequent releases was not available during our research. Therefore, a fit data set and a target data set were derived from the single data set available. Similar to our classification experiments, the fit data set comprised 2/3 of the observations, and the target data set comprised 1/3 of the observations. The models built in our second experiment were validated by cross-validation using the fit data set and by data splitting using the target data (test data) set.

Based on the CBR methodology stated earlier, the following describes the characteristics of our prediction models:

1. A fit data set is specified as the case library. For the first experiment, the fit data set is the entire data set. For the second experiment, the fit data set is 2/3 of the entire data set.

2. A target data set is specified for model validation. For the first experiment, the target data set is the same as the fit data set. For the second experiment, the target data set is 1/3 of the entire data set.

3. A similarity function is selected. We selected Mahalonobis Distance.

4. The number of cases is selected. The model with the optimum results dictates the optimum value for the number of cases, n_N .

5. For prediction models, the solution algorithm is selected. We selected Inverse-Distance Weighted Average.

6. The models are evaluated for accuracy. We used Average Absolute Error (AAE) and Average Relative Error (ARE).

For prediction models, the only parameter to ascertain is n_N. In previous research [22], the optimum model is determined by analyzing the AAE results. By increasing values of n_N, a trend in the AAE values will appear. The AAE values for each succeeding n_N will decrease until they reach a particular n_N value. At that point, the AAE values will begin to increase. The model at this value of n_N, having a minimal AAE is selected as the optimum model.

Prediction Experiment 1 - Entire Data Set: The Mahalonobis Distance similarity function is used to determine the distance from the current case to the cases in the case library. The distance measures are then used in a solution algorithm to obtain the estimated value for the dependent variable. In our case study, we selected the Inverse-Distance Weighted Average solution algorithm because previous research [22], indicates that this solution algorithm has better predictive accuracy than the other solution algorithms available. This research also shows that the Mahalonobis Distance also performs better than the other similarity functions available. Table 13 displays the prediction model results for the entire data set.

The optimum model is determined via Table 13. Following the AAE column, the best model is selected when the AAE values decrease until it reaches its minimum, at which point it starts to increase. This indicates that our optimum model is where n_N equals 13. An AAE value of 1.072 indicates that the model correctly predicted the number of faults discovered in each source file during system test to within 1.072 faults.

Prediction Experiment 2 - Fit/Target Data Set: The prediction experiment involving a fit data set and a target set was validated using cross-validation and data splitting. 50 models were built to ensure the results obtained were due to accurate modeling instead of a fortunate data split. Table 14 displays the results for all 50 models.

The average AAE and ARE values for the models validated using cross-validation are 1.098 and 0.297, respectively. The average AAE and ARE values for the models validated using data splitting are 1.115 and 0.290, respectively. The standard deviations for the models validated using data splitting are similar to those validated using cross-validation. The average n_N value is 10.24. The results show n_N values ranging from 3 to

25. Similar to the classification experiments, this indicates that this parameter should not be fixed at a predetermined value. Instead, n_N should be chosen empirically for each case in order to build an optimum model.

Prediction Experiments- Results: Table 14 summarizes the results from the two prediction experiments conducted. Preliminary research on CBR using City Block Distance showed that CBR provided better accuracy than the corresponding multiple linear regression models [18] for predicting faults [3]. Another study was performed on the use of the Euclidean Distance with CBR classification models [11] which also provided good results when compared to models built using nonparametric discriminant analysis [15]. Subsequent research on CBR has shown that the Mahalonobis Distance yields better performance accuracy with raw data sets than other distance measures such as City-Block Distance and Euclidean Distance [22]. This research provides additional support for the use of CBR as a viable modeling methodology for software quality control. Our results indicate that successful models were built with the independent variables selected. Each experiment revealed that our models are capable of predicting the number of faults discovered in each source file during system test to within one fault.

4 Empirical Case Study 2

This section will first describe the system used for our second empirical study. The different metrics collected will be highlighted and then the results will be presented.

Table 13: Prediction Model Results-Entire Data Set

n_N	Fit (Cross-Validation)	
	AAE	ARE
1	1.078	0.284
3	1.137	0.296
5	1.084	0.288
7	1.077	0.286
9	1.081	0.285
11	1.078	0.282
12	1.075	0.281
13	1.072	0.279
14	1.073	0.279
15	1.08	0.279
17	1.083	0.278
19	1.084	0.277
21	1.084	0.275
23	1.084	0.275
25	1.084	0.274
27	1.085	0.273
29	1.088	0.272
31	1.091	0.272
33	1.093	0.271
35	1.095	0.271
37	1.097	0.27
39	1.098	0.269
41	1.101	0.268
43	1.103	0.268
45	1.105	0.268
47	1.107	0.267
49	1.109	0.267
51	1.111	0.267
53	1.113	0.266
55	1.115	0.266
57	1.117	0.266
59	1.118	0.265
61	1.119	0.264
63	1.12	0.264
65	1.122	0.263

Table 14: Prediction Model Results-Fit/Target Data Set

Data Set	n_N	Fit(Cross Validation)		Target(test)	
		AAE	ARE	AAE	ARE
1	8	1.129	0.31	1.09	0.286
2	7	1.098	0.271	1.008	0.271
3	7	1.158	0.318	1.11	0.318
4	13	1.003	0.299	1.257	0.265
5	9	1.134	0.285	1.013	0.26
6	6	1.139	0.316	1.1	0.317
7	9	1.063	0.309	1.164	0.273
8	20	1.107	0.284	1.133	0.278
9	21	1.096	0.298	1.168	0.281
10	9	1.108	0.296	1.088	0.322
11	9	1.134	0.314	1.03	0.273
12	11	0.966	0.279	1.298	0.272
13	21	1.162	0.297	1.048	0.269
14	12	1.053	0.3	1.233	0.304
15	4	1.046	0.288	1.205	0.338
16	5	1.196	0.312	0.902	0.244
17	15	1.05	0.275	1.175	0.301
18	23	1.238	0.318	0.899	0.24
19	4	0.937	0.266	1.447	0.385
20	18	1.027	0.294	1.25	0.284
21	7	1.076	0.3	1.198	0.309
22	9	0.987	0.28	1.22	0.262
23	5	1.077	0.29	1.195	0.326
24	10	1.147	0.28	1.005	0.306
25	5	1.126	0.326	1.051	0.259
26	4	1.115	0.266	0.99	0.309
27	9	0.99	0.302	1.288	0.263
28	11	1.143	0.29	0.981	0.292
29	9	1.103	0.304	1.185	0.294
30	3	1.046	0.318	1.163	0.289
31	14	1.19	0.3	1.02	0.287
32	7	1.174	0.309	0.912	0.232
33	23	1.071	0.274	1.293	0.319
34	14	1.151	0.295	0.971	0.263
35	8	1.19	0.324	0.975	0.265
36	8	1.12	0.288	1.007	0.265
37	8	1.052	0.297	1.252	0.335
38	7	1.096	0.302	0.989	0.3
39	11	1.163	0.296	1.038	0.31
40	9	0.996	0.299	1.263	0.308
41	25	1.21	0.303	1.004	0.255
42	5	1.037	0.287	1.285	0.354
43	9	1.091	0.304	1.123	0.288
44	7	1.154	0.304	0.981	0.288
45	8	1.134	0.305	1.055	0.269
46	8	1.157	0.287	0.958	0.308
47	5	0.979	0.302	1.342	0.282
48	15	1.18	0.3	0.932	0.289
49	9	0.944	0.307	1.478	0.303
50	9	1.145	0.281	0.968	0.292
Average	10.24	1.098	0.297	1.115	0.29
STD	5.44	0.072	0.014	0.141	0.029

Table 14: Prediction Model Result

Cross validation(Entire Data Set)		Cross validation(Fit)		Data Splitting(Fit)		Average # of cases
AAE	ARE	AAE	ARE	AAE	ARE	n_N
1.072	0.279	1.098	0.297	1.115	0.290	10.24

4.1 System description

The system is exactly the same as the one used and presented in the previous section describing our first case study.

4.2 Data Collection Effort in Case Study 2

For our second case study regarding the fault removal process, we primarily used the problem reporting database to obtain a list of the faults discovered during system test. Through this database, we were able to directly obtain most of the metrics used in our study. The only exception was PR_FIX_LOC, which is the number of lines of code for a collection of files before a fault-fix. This metric was not captured directly in the database. Therefore, an internally developed line count tool was used to determine this information.

Once the data selection process was complete, the data was preprocessed and cleaned. This step is necessary to remove any outliers or illogical data points. An outlier is an instance which values are by far out of the usual range for some variables.

Table 15 gives the list of product and process metrics used in our second case study.

Table 15: Product and Process Metrics for Case Study 2

Product Metrics	Description
FILES_FIX	Number of source files modified to fix a fault.
PR_FIX_LOC	Number of lines of code, before the fix, for the collection of source files modified to fix the fault.
CODE_CHRN	Number of lines of code modified by the developer within the collection of source files modified to fix. This is the summation of the lines added, deleted or modified.
PO_FIX_LOC	Number of lines of code, after the fix, for the collection of source files modified to fix the fault.
Process Metrics	**Description**
FIX_HRS	Time, in hours, spent by developers to fix the fault.

4.3 Case Study 2 Description

Case Study 2 analyzes our second data set that contains data on faults found during system test. After the data set was preprocessed and cleaned, 230 observations remained. In this particular case study, an observation is a fault discovered during system test and its associated fault removal metrics. One process metric and four product metrics were used as independent variables. The five metrics are described in Table 15. Once again, data reduction or transformation techniques were not necessary. This set of metrics will be referred to as RAW_FAULT_FIXES-5. The correlations among the five independent variables are shown in table 16. The dependent variable in this second case study is the

Table 16: Correlation Statistics for $RAW\_FILES - 5$

CORRELATION	FIX_HRS	PR_FIX_LOC	CODE_CHRN	PO_FIX_LOC	FILES_FIX
FIX_HRS	1.00	0.1674	0.1193	0.1696	0.1605
PR_FIX_LOC		1.00	0.3821	0.9996	0.3842
CODE_CHRN			1.00	0.3943	0.3783
PO_FIX_LOC				1.00	0.3884
FILES_FIX					1.00

outcome of a fault-fix inspection.

While the first case study seeks to identify fault-prone areas to guide reliability enhancement efforts, this second case study analyzes the effectiveness of the organization's fault removal process to provide an opportunity for resource allocation improvements.

4.4 Case Study 2 Methodology

This second case study involved building a software quality classification model to predict the outcome of a fault-fix inspection. This allows managers the opportunity to assess the effectiveness of their organization's fault removal process. There are many reasons why managers would want to improve their software processes and quality. These include a moral obligation, customer satisfaction, cost effectiveness, predictability, application demand, and international competition. In particular, the cost of labor has become a major factor in software development costs. By improving the development processes, managers can make more efficient use of personnel resources. This classification model was validated using cross-validation. Because the data set was relatively small, validation by data splitting was not appropriate.

The following steps describe the modeling methodology used in our second case study for our classification model:

1. Collect configuration management and problem reporting data from a past project.

2. Preprocess and clean the data.

3. Identify the following: the list of independent variables and the dependent variable. We identified five independent variables, and our dependent variable is the outcome of a fault-fix inspection.

4. For classification modeling: Determine the class of each observation. The classification is as follows:

   $$Class = \begin{cases} REJECT, & \text{if Outcome} = 1 \\ ACCEPT, & \text{otherwise} \end{cases} \tag{20}$$

   ACCEPT will accept the inspection for the fault-fix, while *REJECT* will require re-work to be performed. An *Outcome* equal to 1 indicates that the inspection was necessary in order to reveal issues with the fault-fix.

5. Prepare the fit and target data (test data) sets. A fit data set is used to build the model, and a target data set is used to validate the model.

6. Select a modeling methodology and build the model. We used CBR. The models were validated and evaluated for their accuracy. We validated the models using cross-validation.

4.5 Classification Experiments

Based on the CBR methodology described in section 2, the following are the characteristics of our classification model for Case Study 2:

1. A fit data set is specified as the case library. The fit data set is the entire data set of faults discovered by system test.

2. A target data set is specified for model validation. The target data set is the same as the fit data set.

3. A similarity function is selected. We selected Mahalonobis Distance.

4. The number of cases is selected. The model with the optimum results dictates the optimum value for the number of cases, n_N.

5. For classification models, the classification method is selected. We selected Data Clustering.

6. The models are evaluated for accuracy. We used Type I and Type II misclassification rates.

In addition, the cost ratio parameter was varied, and the cost ratio that yielded the most balanced misclassification rates was selected for each n_N. When the previously stated classification is used, 19/230 fault-fixes or 8.26% are classified as $REJECT$. The remaining 211/230 fault-fixes or 91.74% are classified as $ACCEPT$. The same methodology used in the first case study for the classification experiments was followed. Table 17 identifies the cost ratios and their associated Type I and Type II misclassification rates where $n_N = 6$. Figure 5 displays the tradeoff between the misclassification rates as a

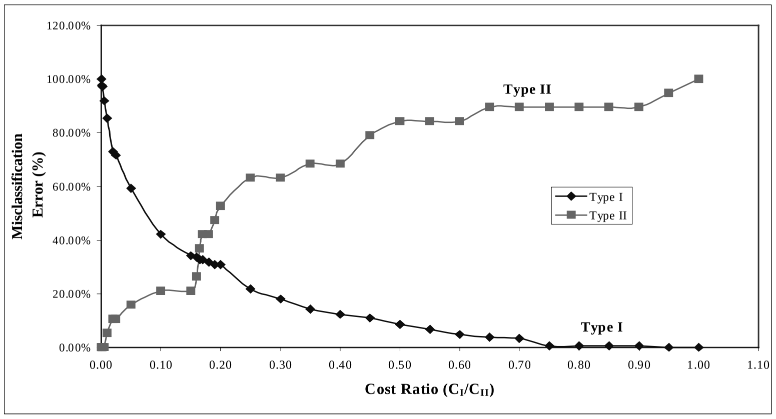

Figure 5: Type I and Type II Misclassification Rates Per Cost Ratio- Threshold 2

function of the cost ratio. Table 18 displays the model results for this case study. Table 17 indicates that for $n_N = 6$, the optimum cost ratio is 0.15. Table 18 indicates that the optimum model is $n_N = 6$ and the cost ratio = 0.15.

The Type I and Type II misclassification rates are 34.12% and 21.05%, respectively. Following the same analysis as in the first case study, and given a cost ratio of 0.15, a Type I misclassification cost would be one unit, and a Type II misclassification cost would be 6.67 (1/0.15) units. Therefore, the actual cost of a Type II misclassification would be 7.67, which is 6.67 plus the one unit cost of extra reviews. A perfect model would cost 19 units for review and yield 146 (19*7.67) units in avoided inspection costs. When this perfect model is compared to the model built, the expected cost of misclassifications would be 98 (211*0.3412*1)+(19*0.2105*(7.67-1)) units. The cost of reviews would be 87 (211*0.3412*1) + (19*0.7895*1) units. The benefit would be reliability improvement for 15 (19*0.7895) fault-fixes, which might avoid 115 (15*7.67) units in inspection costs later. The ROI is 115:87.

Table 19 provides the evaluation results for the optimum model. Using this model, 15/19 fault-fix inspection outcomes are correctly classified as $REJECT$. Similarly, 139/211 fault-fix inspection outcomes are correctly classified as $ACCEPT$. From this model, we would expect to classify 87 fault-fix inspection outcomes as $REJECT$. If extra effort is invested to improve the reliability of these 87 fault-fixes, then the fraction of the inspections classified as $REJECT$ could be reduced from 19/230 or 8.26% to 1.74% (0.0826 * 0.2105).

4.6 Classification Experiments- Results

If managers were to implement this classification model, they could expect to save 24% ((115-87)/115) in inspection costs. The current fault removal process provides independent verification by system test for all faults logged. As a result, even if a model had a somewhat unsatisfying Type II misclassification rate, the current process would catch the fault-fixes that should have been inspected. Therefore, quality models should be used to allow a more efficient use of time and resources. Also, it has been proven that the cost of diagnosing and correcting faults late in development is much greater than when finding the faults early [14]. Therefore, the use of software quality models can be quite advantageous in this respect.

Using this model, 139 fault-fix inspections that resulted in an outcome of $ACCEPT$, were correctly classified. This indicates that managers would be able to save time and resources by not performing 139 inspections, which would have yielded no errors. 15 inspections would be performed, and these 15 would reveal errors in the fault-fixes. In addition, 72 fault-fix inspections would have been conducted on fault-fixes that should

Table 17: Cost Ratio Misclassification where $n_N = 6$.

n_N	Fit (Cross-Validation)	
	AAE	ARE
0.0005	100.00%	0.00%
0.001	97.63%	0.00%
0.002	97.16%	0.00%
0.0025	97.16%	0.00%
0.005	91.94%	0.00%
0.01	85.31%	5.26%
0.02	72.99%	10.53%
0.025	71.56%	10.53%
0.05	59.24%	15.79%
0.1	42.18%	21.05%
0.15	**34.12%**	**21.05%**
0.16	33.65%	26.32%
0.165	32.70%	36.84%
0.17	32.70%	42.11%
0.18	31.75%	42.11%
0.19	30.81%	47.37%
0.2	30.81%	52.63%
0.25	21.80%	63.16%
0.3	18.01%	63.16%
0.35	14.22%	68.42%
0.4	12.32%	68.42%
0.45	10.90%	78.95%
0.5	8.53%	84.21%
0.55	6.64%	84.21%
0.6	4.74%	84.21%
0.65	3.79%	89.47%
0.7	3.32%	89.47%
0.75	0.47%	89.47%
0.8	0.47%	89.47%
0.85	0.47%	89.47%
0.9	0.47%	89.47%
0.95	0.00%	94.74%
1	0.00%	100.00%

Table 18: Classification Model Results-Entire Data Set

n_N	CI/CII	Fit(Cross Validation)	
		Type I	Type II
1	0.25	41.23%	26.32%
2	0.2	39.34%	21.05%
3	0.15	45.02%	21.05%
4	0.15	39.81%	21.05%
5	0.15	37.44%	21.05%
6	**0.15**	**34.12%**	**21.05%**
7	0.1	40.76%	21.05%
8	0.1	37.92%	31.58%
9	0.1	34.60%	36.84%
10	0.1	33.18%	42.11%
11	0.1	27.01%	36.84%
12	0.1	24.17%	42.11%
13	0.1	22.28%	42.11%
14	0.1	21.33%	42.11%
15	0.1	20.85%	47.37%
16	0.1	18.48%	52.63%
17	0.1	17.54%	78.95%
18	0.1	7.58%	78.95%
19	0.1	5.69%	100.00%
20	0.1	0.00%	100.00%
21	0.1	0.00%	100.00%
22	0.1	0.00%	100.00%
23	0.1	0.00%	100.00%
24	0.1	0.00%	100.00%
25	0.1	0.00%	100.00%

have been accepted. While this does result in wasted efforts, it is still a significant improvement from having to perform inspections for all 211 fault-fixes, which would have yielded no errors. Four fault-fix inspections were classified as $ACCEPT$ when they should have been classified as $REJECT$. However, system test verifies all fault-fixes in subsequent releases. Therefore, the release of four faults that were not fixed properly,

Table 19: Evaluation of Entire Data Set

Model (Cross-Validation)			
Actual	ACCEPT	REJECT	TOTAL
ACCEPT	139 65.88%	72 34.12%	211 100%
REJECT	4 21.05%	15 78.95%	19 100%
TOTAL	143	87	230
PERCENT	62.17%	37.83%	100%
PRIOR	91.74%	8.26%	

to system test, is a small tradeoff for resources saved by not having to perform 139 unnecessary inspections.

In our data set, the average time spent by an inspection team, preparing and conducting inspections is 0.79 of an hour. The time savings for the 139 inspections would then be 109 (139*0.79) hours. More productive use of these hours could be spent on the riskier areas of the project to provide greater reliability instead of wasting them on ineffective inspections of fault-fixes. Given additional data sets with more observations, the resource allocation improvements would no doubt have a great impact on the overall quality of the software. As the delivery date for a product quickly approaches, time becomes an increasingly precious commodity. The use of the software quality models built can allow management to make more appropriate use of the precious time left. This will ensure that the established reliability and quality standards can still be achieved in the presence of tight schedules for product delivery to market.

5 Conclusions

This section draws conclusions from the results of the experiments discussed in the previous section. It also provides the areas for future work.

With the number of software driven devices on the rise, and more people depending on them, it is imperative for organizations to ensure the reliability of its software systems. However, the costs associated with developing software are also ever increasing. It then becomes difficult for project managers to remain within budget and on schedule. As a result, quality and reliability are often reduced in order to meet the promised delivery dates.

Through the use of software quality models, managers have the opportunity to discover areas for improvement in their development processes. The models can provide managers with the insight that will allow them to improve their reliability enhancement and resource allocation strategies.

From our research, we have proven that CBR is a simple modeling methodology that can be used to develop accurate and useful models faster, easier, and cheaper. It is fast because very few metrics were needed to build these models. In both of our case studies, only five independent variables were needed. It is easy because CBR is simple, and easy-to-use and interpret. The CBR methodology is a methodology to which any user can relate. In addition, CBR works very well with raw metrics. The combination of these reasons makes CBR very cheap to implement.

The first case study proves that accurate software quality models can be built to predict the quality of the software prior to system test. The classification models indicated a ROI for determining the FP files early at approximately 50%. This reveals that by using classification models, managers have the opportunity to save time and money by identifying FP files early. The prediction models were much more impressive because they were able to predict the number of faults discovered in each source file during system test to within one fault. By having this information early, test managers will know exactly which files are more likely to contain faults. They can also ascertain how long to continue testing by comparing the number of faults discovered to the predicted number of faults. As a result, resource allocation during system test could be greatly enhanced.

The second case study proves that a software quality model can be built to determine the outcome of a fault-fix inspection, thereby allowing managers to save time, money, and energy by inspecting only the fault-fixes that contain errors. The ROI for determining the outcome of the fault-fix inspections early was approximately 24%. This indicates that there is a potential opportunity to forego the ineffective fault-fix inspections for more productive tasks.

Software quality models can be of great use to managers. They have the ability to provide management with insight to better control their development and testing processes. By making their processes more efficient and effective, managers have the ability to provide a reliable, high quality product in less time.

The use of CBR with few raw metrics revealed interesting results. However, it is necessary to perform more investigation on subsequent releases or projects using CBR to further establish the benefits of CBR as an efficient modeling technique. Other product and process metrics such as call graph and control flow graph metrics mentioned in section 2 could be used to determine their effectiveness with CBR. In addition, larger data sets for fault-fixes could be obtained to further validate the impact of software quality models on resource allocation strategies.

Acknowledgments

We thank Ken McGill for his encouragement and support. We thank Bojan Cukic for helpful discussions. This work was supported in part by Cooperative Agreement NCC 2-1141 from NASA Ames Research Center, Software Technology Division (Independent Verification and Validation Facility). The findings and opinions in this paper belong solely to the authors, and are not necessarily those of the sponsor. Special thanks to Naeem Seliya for his patient reviews of this report.

References

[1] Y. Berkovich. Software quality prediction using case-based reasoning. Master's thesis, Florida Atlantic University, Boca Raton, Florida USA, Aug. 2000. Advised by Taghi M. Khoshgoftaar.

[2] T. M. Khoshgoftaar, E. B. Allen,Y. Berkovich, F. D. Ross. Diagnostic Tools for indentifying high-risk software modules: a case-based reasoning approach. Submitted to NASA Independent Verification and Validation Facility. Technical Report TR-CSE-00-20, Florida Atlantic University, Boca Raton, Florida USA, June 2000.

[3] K. Ganesan, T. M. Khoshgoftaar, and E. B. Allen. Case-based software quality prediction. *International Journal of Software Engineering and Knowlegde Engineering*, 10(2):139–152, 2000.

[4] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Early quality prediction: A case study in telecommunications. *IEEE Software*, 13(1):65–71, Jan. 1996.

[5] T. M. Khoshgoftaar and E. B. Allen. Modeling the risk of software faults. Technical Report TR-CSE-00-06, Florida Atlantic University, Boca Raton, Florida USA, Feb. 2000.

[6] T. M. Khoshgoftaar and E. B. Allen. Predicting the order of fault-prone modules in legacy software. In *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, pages 344–353, Paderborn, Germany, Nov. 1998. IEEE Computer Society.

[7] T. M. Khoshgoftaar, E. B. Allen, L. A. Bullard, R. Halstead, and G. P. Trio. A tree-based classification model for analysis of a military software system. In *Proceedings of the IEEE High-Assurance Systems Engineering Workshop*, pages 244–251, Niagara on the Lake, Ontario, Canada, Oct. 1996. IEEE Computer Society.

[8] T. M. Khoshgoftaar, D. L. Lanning, and A. S. Pandya. A comparative study of pattern recognition techniques for quality evaluation of telecommunications software. *IEEE Journal on Selected Areas in Communications*, 12(2):279–291, Feb. 1994.

[9] T. M. Khoshgoftaar and R. M. Szabo. Improving code churn predictions during the system test and maintenance phases. In *Proceedings of the International Conference on Software Maintenance*, pages 58–67, Victoria, BC Canada, Sept. 1994. IEEE Computer Society.

[10] T. M. Khoshgoftaar, E. B. Allen, and J. C. Busboom. Software quality modeling: The software measurement analysis and reliability toolkit. *IEEE International Conference on Tools with Artificial Intelligence*, pages 54–61, Nov 2000.

[11] T. M. Khoshgoftaar, K. Ganesan, E. B. Allen, F. D. Ross, R. Munikoti, N. Goel, and A. Nandi. Predicting fault-prone modules with case-based reasoning. In *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, pages 27–35, Albuquerque, New Mexico USA, Nov. 1997. IEEE Computer Society.

[12] T. M. Khoshgoftaar and E. B. Allen. A practical classification rule for software quality models. *IEEE Transactions on Reliability*, 49(2), June 2000. In press.

[13] T. M. Khoshgoftaar and E. B. Allen and W. D. Jones and J. P. Hudepohl. Data Mining for Predictiors of Software Quality *International Journal of Software Engineering and Knowledge Engineering*, 1999.

[14] T. M. Khoshgoftaar and E. B. Allen Classification of Fault-Prone Software Modules: Prior Probabilities, Costs, and Model Evaluation *Empirical Software Engineering: An International Journal*, Volume3:275–298, Sep. 1998.

[15] T. M. Khoshgoftaar, E. B. Allen, N. Goel, A. Nandi and J. McMullan Detection of Software Modules with High Debug Code Churn in a Very Large Legacy System *Proceedings of the Seventh International Symposium on Software Reliability Engineering*, pages 364–371, Oct. 1996.

[16] R. Kowalski. AI and software engineering. In *Artificial Intelligence and Software Engineering*, pages 339–352. Ablex Publishing, Norwood, NJ USA, 1991.

[17] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE–2(4):308–320, Dec. 1976.

[18] G. J. Myers. *Composite/Structured Design*. Van Nostrand Reinhold, New York, 1978.

[19] S. L. Pfleeger. Assessing measurement. *IEEE Software*, 14(2):25–26, Mar. 1997. Editor's introduction to special issue.

[20] A. A. Porter and R. W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Software*, 7(2):46–54, Mar. 1990.

[21] F. Ross. An Empirical Study of Analogy Based Software Quality Classification Models. Master's thesis, FAU, May 2001.

[22] N. Sundaresh. An Empirical Study of Analogy Based Software Fault Prediction. Master's thesis, FAU, May 2001.

[23] Z. Xu Fuzzy Logic Techniques for Software Reliability Engineering. Ph.D. dissertation, FAU, May 2001. Advised by T. M. Khoshgoftaar.